US006549988B1

(12) **United States Patent**
Gertner

(10) **Patent No.: US 6,549,988 B1**
(45) **Date of Patent: Apr. 15, 2003**

(54) **DATA STORAGE SYSTEM COMPRISING A NETWORK OF PCS AND METHOD USING SAME**

(76) Inventor: **Ilya Gertner**, 5 Gasuigut La., Framingham, MA (US) 01701

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/236,409**

(22) Filed: **Jan. 22, 1999**

(51) **Int. Cl.**[7] **G06F 17/30**
(52) **U.S. Cl.** **711/141**; 711/142; 711/143; 711/147; 711/148
(58) **Field of Search** 711/3, 117–126, 711/141–148, 152

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,577,226	A	*	11/1996	Percival	711/119
5,600,817	A		2/1997	Macon, Jr. et al.	
5,644,751	A		7/1997	Burnett et al.	
5,649,152	A		7/1997	Obran et al.	
5,701,516	A		12/1997	Chen et al.	
5,715,455	A		2/1998	Macon, Jr. et al.	
5,717,884	A		2/1998	Gzym et al.	
5,742,792	A		4/1998	Yandi et al.	
5,743,933	A		4/1998	Kijima et al.	
5,748,985	A		5/1998	Kanai et al.	
5,751,993	A		5/1998	Ofek et al.	
5,758,050	A		5/1998	Brady et al.	
5,787,473	A		7/1998	Ofek et al.	
5,790,795	A		8/1998	Hough	
5,802,553	A		9/1998	Robinson et al.	
5,805,857	A		9/1998	Colegrove	
5,819,292	A		10/1998	Hitz et al.	
5,819,310	A		10/1998	Vishlitzky et al.	
5,828,475	A		10/1998	Bennet et al.	
5,841,997	A		11/1998	Bleiwess et al.	
5,848,251	A		12/1998	Lomelino et al.	
5,852,715	A		12/1998	Razid et al.	
5,854,942	A		12/1998	Penokie	
5,860,026	A		1/1999	Kitta et al.	
5,860,137	A		1/1999	Raz et al.	
5,887,146	A	*	3/1999	Baxter et al.	710/104
6,026,461	A	*	2/2000	Baxter et al.	710/244
6,044,438	A	*	3/2000	Olnowich	711/130
6,122,659	A	*	9/2000	Olnowich	709/213

OTHER PUBLICATIONS

Smith, Cache Memories, Computer Surveys, vol. 14, No. 3 Sep. 1982 (Research paper).
Karedla, et al., Caching Strategies to Improve Disk System Performance, Computer, vol. 27, No. 3, Mar. 1994 (Research paper).
Neema, Data Sharing, Storage Management Solutions, vol. 3, No. 3, May 1998.
Hoetger, Jerry, Storage Management in UNIX environments storage management solutions, vol. 3, No. 4, Aug. 1998.

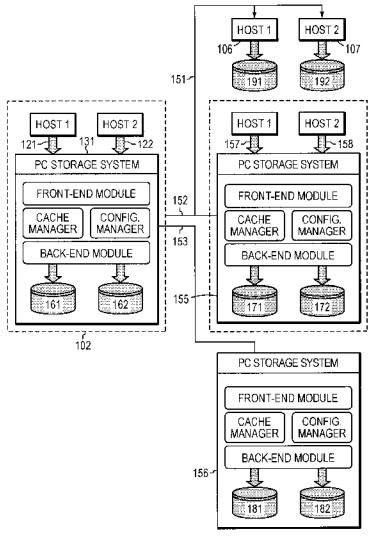
* cited by examiner

Primary Examiner—T. U. Nguyen
(74) *Attorney, Agent, or Firm*—Testa, Hurwitz & Thibeault, LLP

(57) **ABSTRACT**

A data storage system comprising a network of PCs each of which includes a cache memory, I/O channel adapter for transmitting data over the channel and a network adapter for transmitting control signals and data over the network. In one embodiment, a method for managing resources in a cache manager ensures consistency of data stored in the distributed cache. In another embodiment, a method for sharing data between two or more heterogeneous hosts including the steps of: reading a record in a format compatible with one computer; identifying a translation module with the second computer; translating the record into a format compatible with the second computer and writing said translated record into a cache memory.

6 Claims, 13 Drawing Sheets



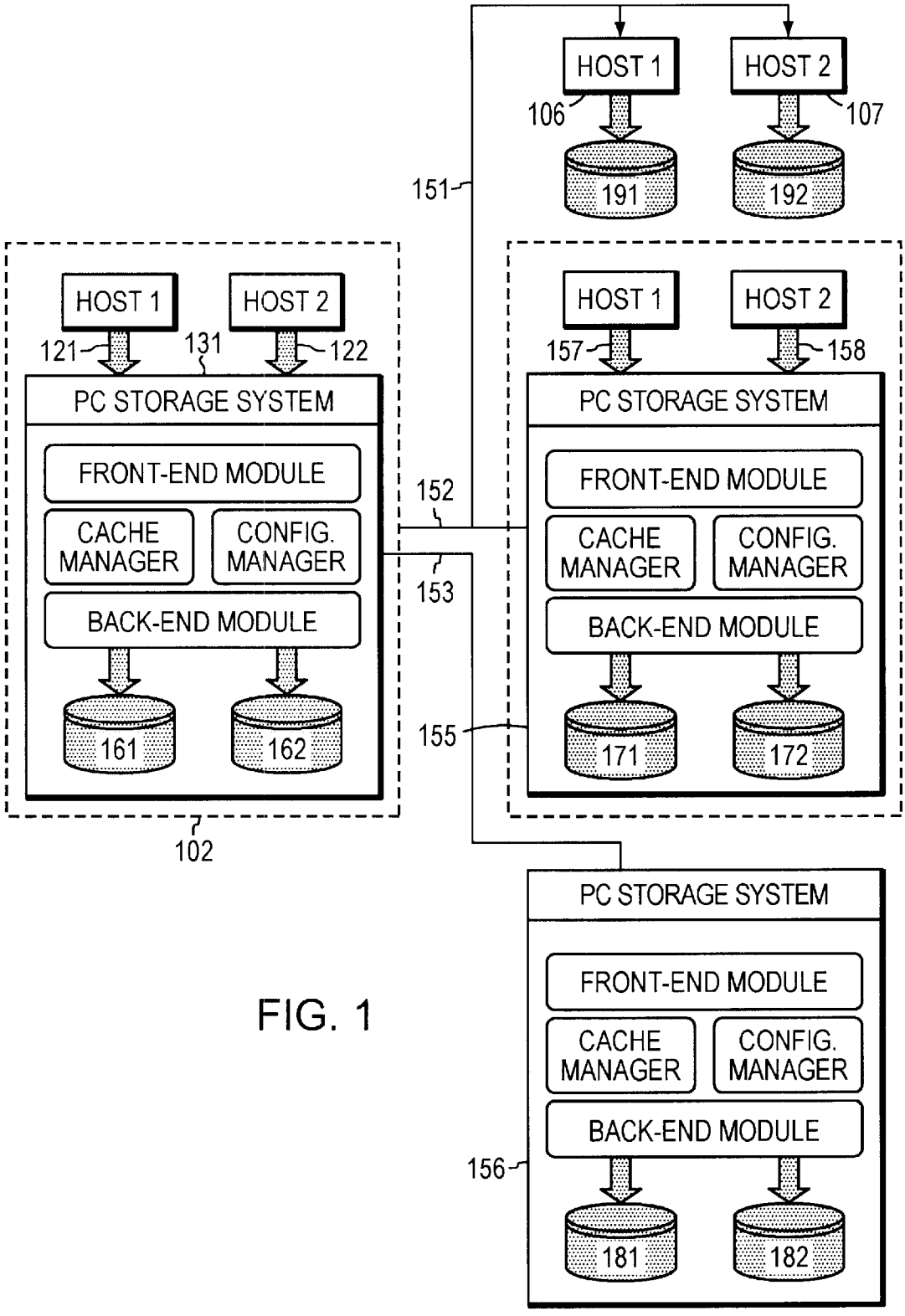

HOST 1
HOST 2
106
107
191
192
151
HOST 1
HOST 2
121
131
122
PC STORAGE SYSTEM
FRONT-END MODULE
CACHE MANAGER
CONFIG. MANAGER
BACK-END MODULE
161
162
102
152
153
HOST 1
HOST 2
157
158
PC STORAGE SYSTEM
FRONT-END MODULE
CACHE MANAGER
CONFIG. MANAGER
BACK-END MODULE
171
172
155
PC STORAGE SYSTEM
FRONT-END MODULE
CACHE MANAGER
CONFIG. MANAGER
BACK-END MODULE
156
181
182

FIG. 1

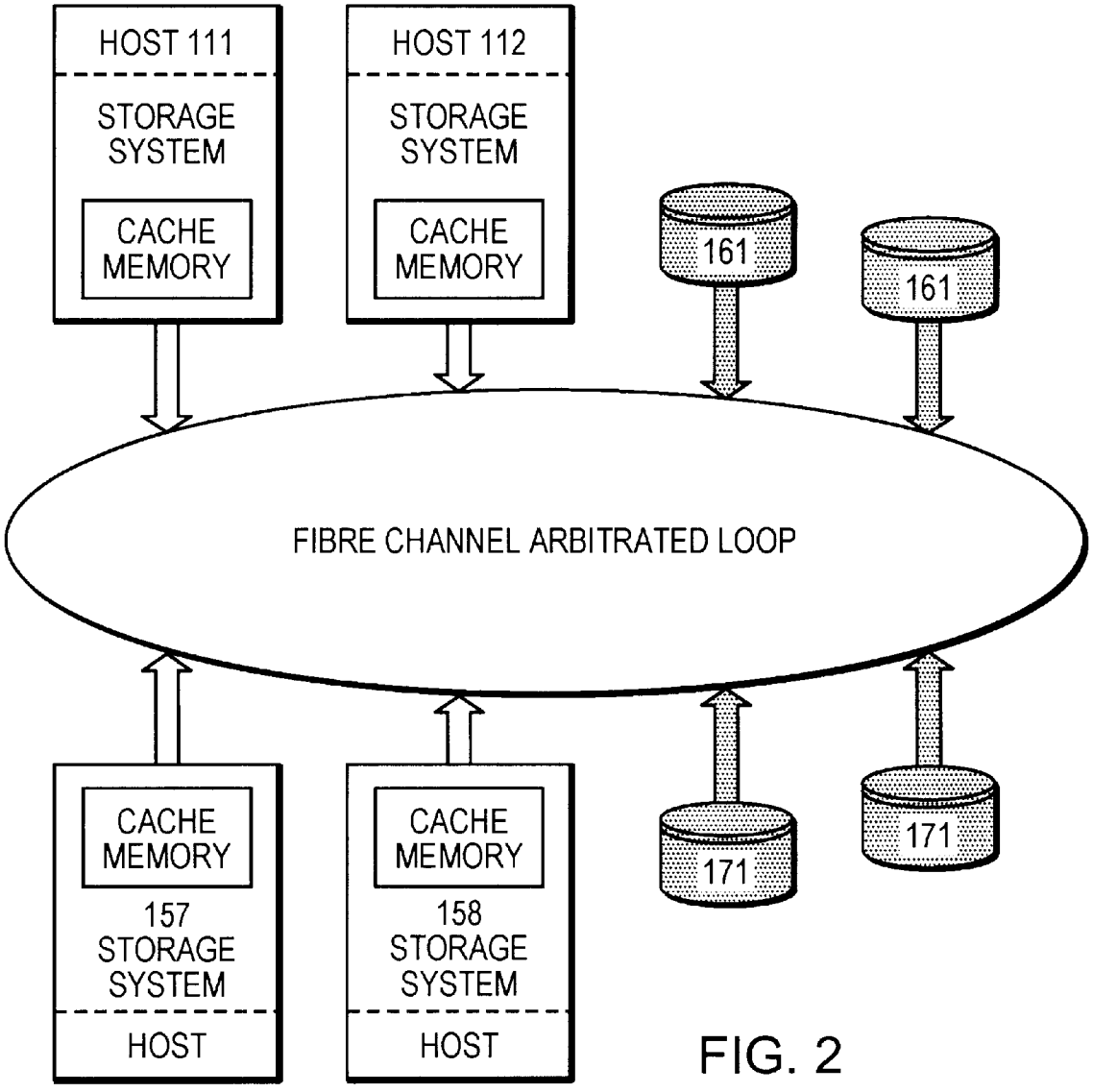

HOST 111
STORAGE SYSTEM
CACHE MEMORY
HOST 112
STORAGE SYSTEM
CACHE MEMORY
161
161
FIBRE CHANNEL ARBITRATED LOOP
CACHE MEMORY
157 STORAGE SYSTEM
HOST
CACHE MEMORY
158 STORAGE SYSTEM
HOST
171
171

FIG. 2

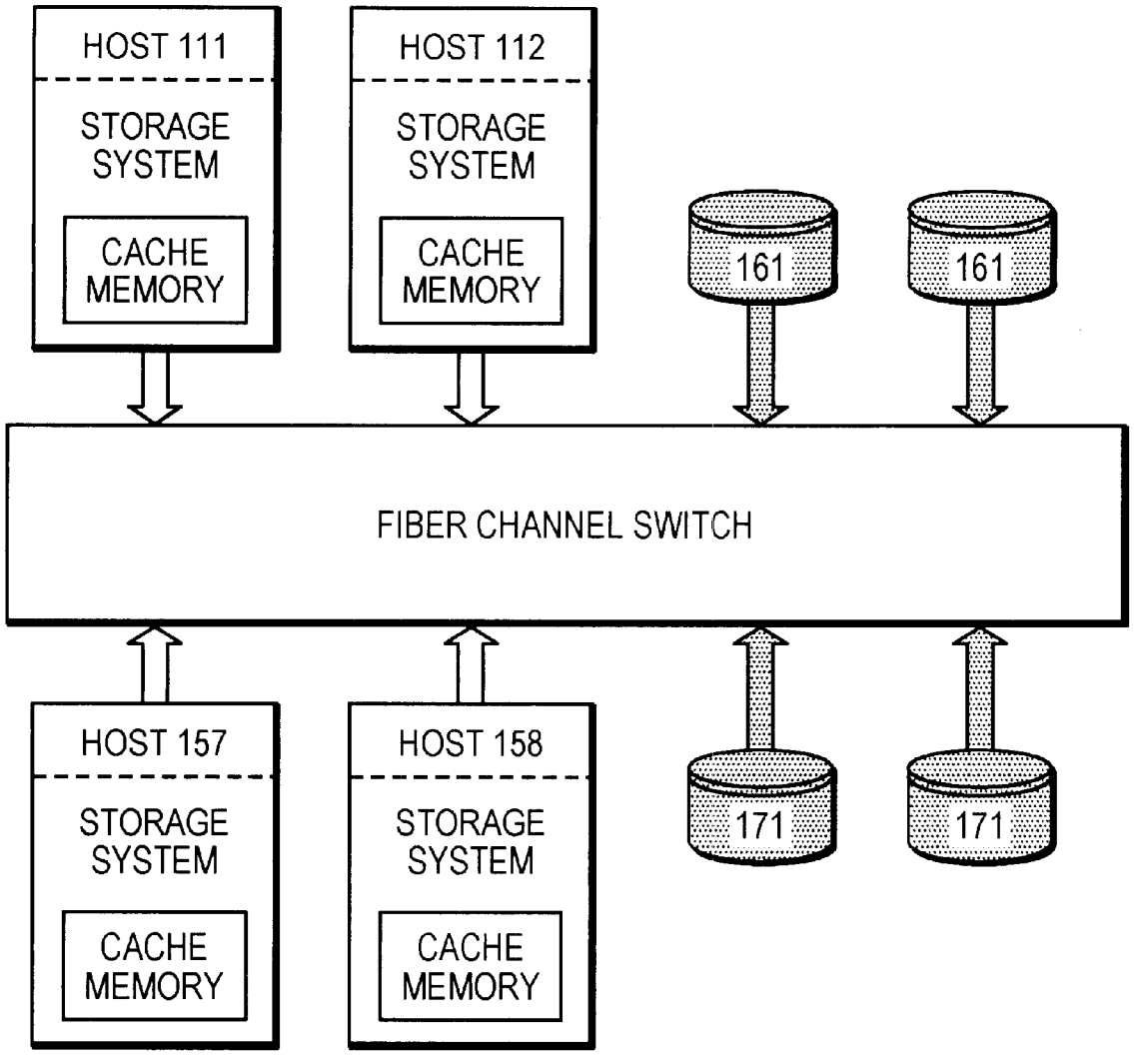

HOST 111
STORAGE SYSTEM
CACHE MEMORY
HOST 112
STORAGE SYSTEM
CACHE MEMORY
161
161
FIBER CHANNEL SWITCH
HOST 157
STORAGE SYSTEM
CACHE MEMORY
HOST 158
STORAGE SYSTEM
CACHE MEMORY
171
171

FIG. 2A

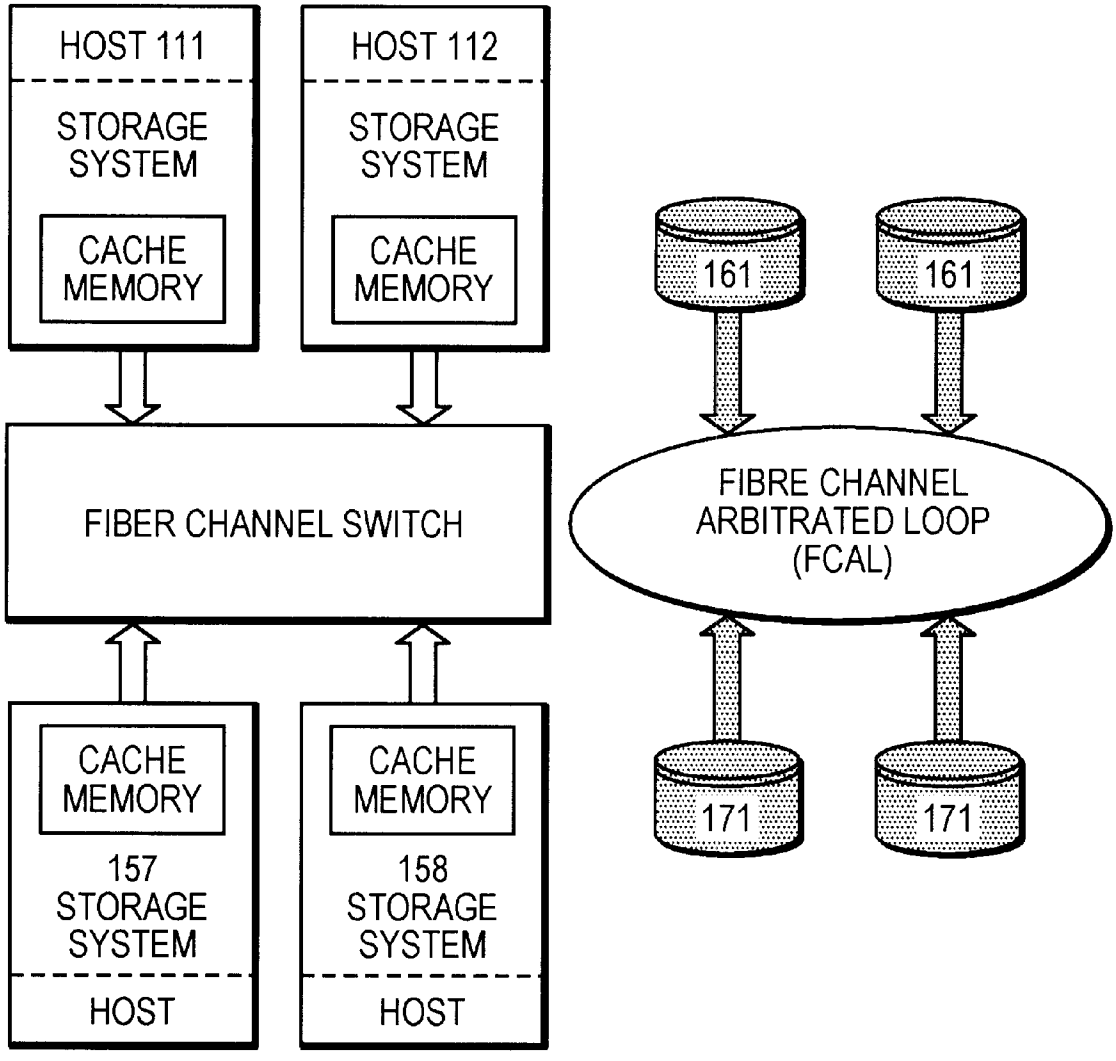

HOST 111
STORAGE SYSTEM
CACHE MEMORY
HOST 112
STORAGE SYSTEM
CACHE MEMORY
161
161
FIBRE CHANNEL ARBITRATED LOOP (FCAL)
FIBER CHANNEL SWITCH
CACHE MEMORY
157 STORAGE SYSTEM
HOST
CACHE MEMORY
158 STORAGE SYSTEM
HOST
171
171

FIG. 2B

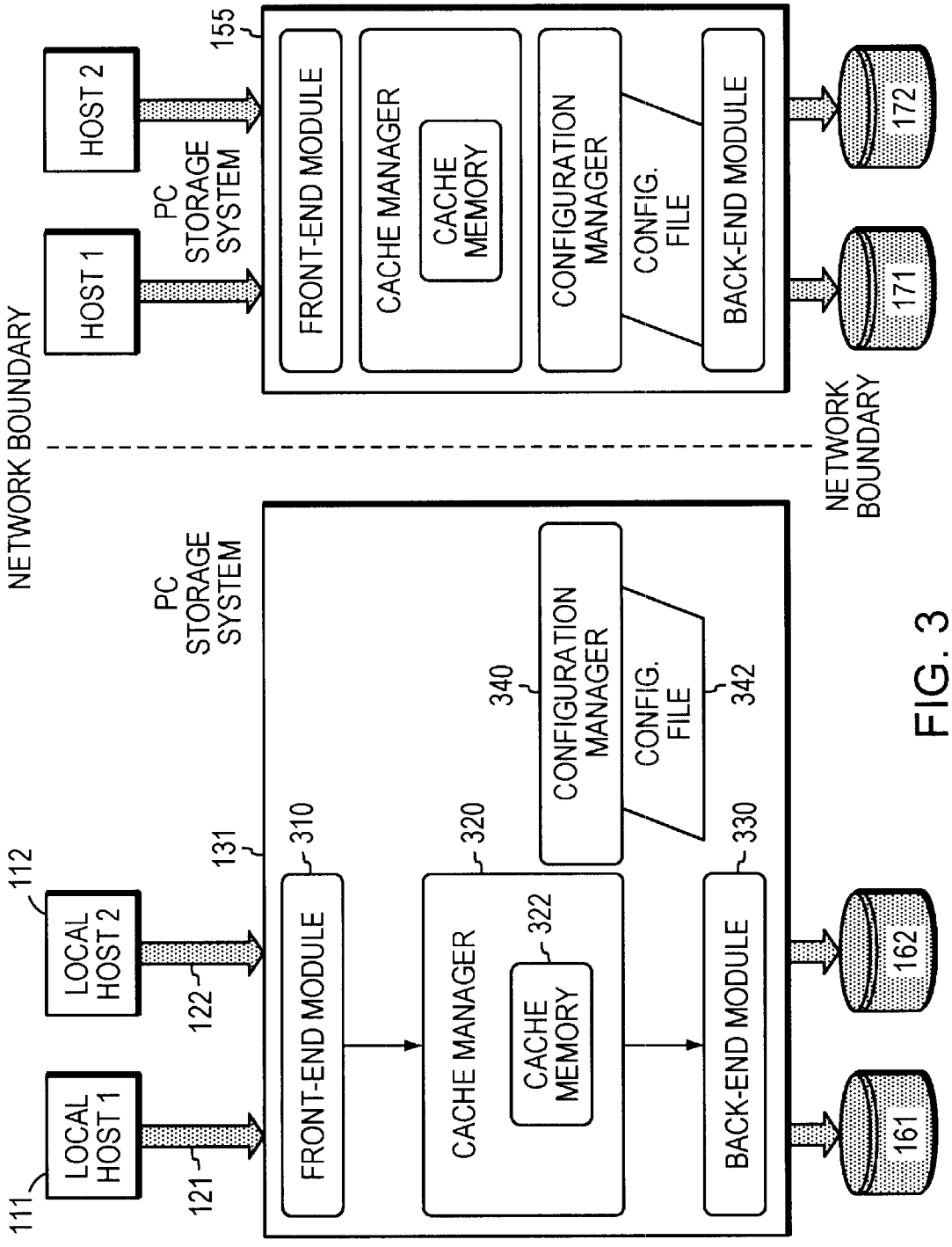

NETWORK BOUNDARY
HOST 1
HOST 2
PC STORAGE SYSTEM
155
FRONT-END MODULE
CACHE MANAGER
CACHE MEMORY
CONFIGURATION MANAGER
CONFIG. FILE
BACK-END MODULE
172
171
NETWORK BOUNDARY
PC STORAGE SYSTEM
LOCAL HOST 1
LOCAL HOST 2
111
112
121
122
131
310
320
322
330
340
342
FRONT-END MODULE
CACHE MANAGER
CACHE MEMORY
CONFIGURATION MANAGER
CONFIG. FILE
BACK-END MODULE
161
162

FIG. 3

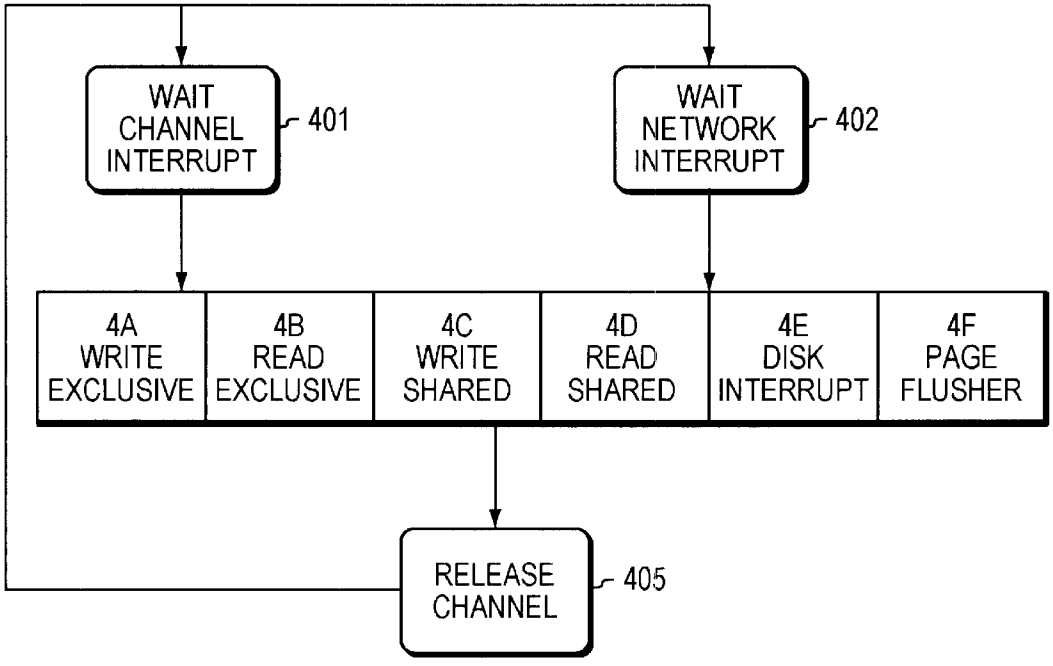
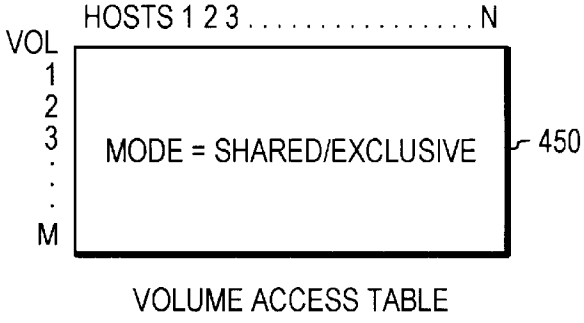


FIG. 4



VOLUME ACCESS TABLE

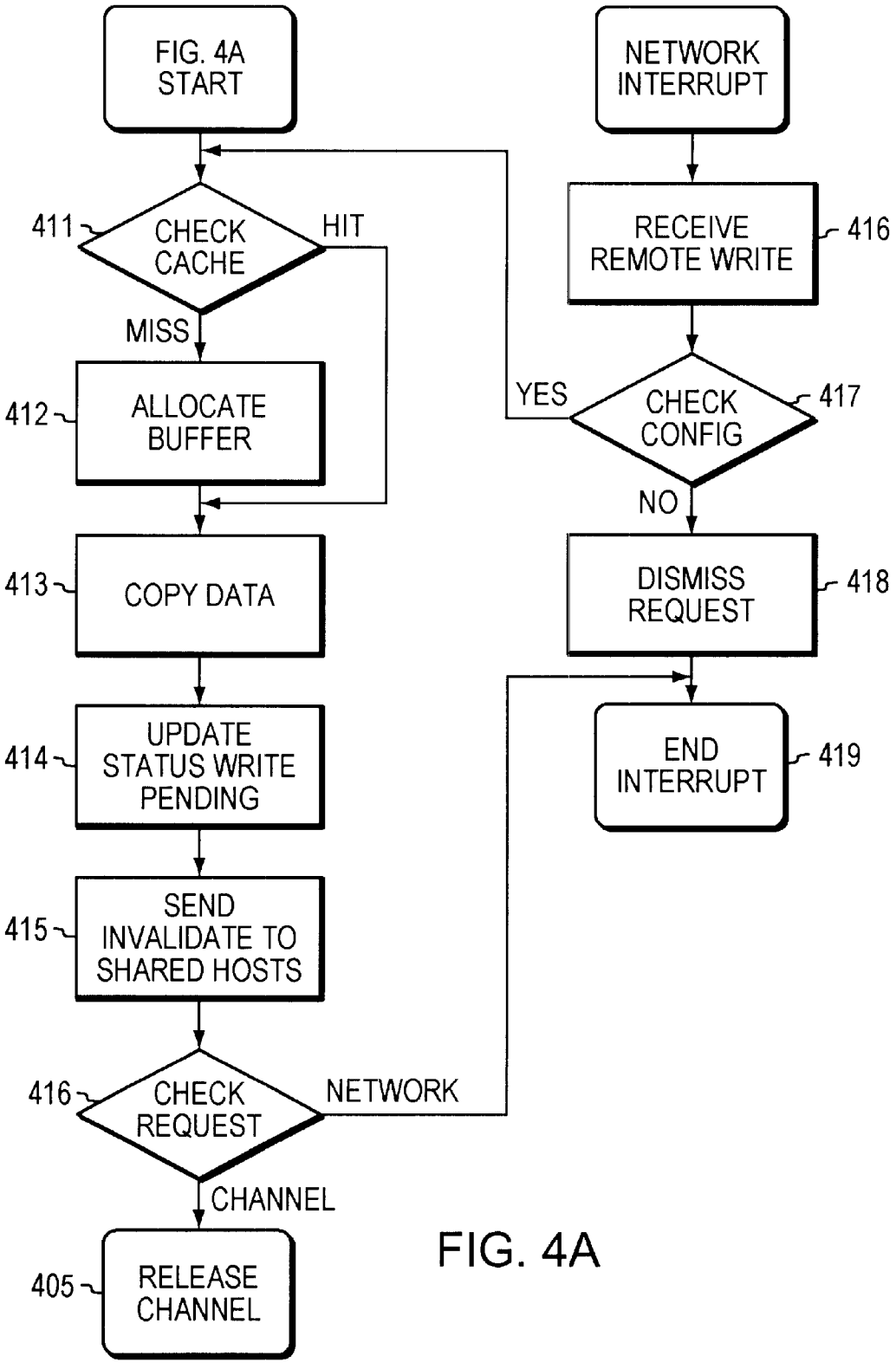


FIG. 4A START
411
CHECK CACHE
HIT
MISS
412
ALLOCATE BUFFER
413
COPY DATA
414
UPDATE STATUS WRITE PENDING
415
SEND INVALIDATE TO SHARED HOSTS
416
CHECK REQUEST
NETWORK
CHANNEL
405
RELEASE CHANNEL
NETWORK INTERRUPT
RECEIVE REMOTE WRITE
416
YES
CHECK CONFIG
417
NO
DISMISS REQUEST
418
END INTERRUPT
419

FIG. 4A

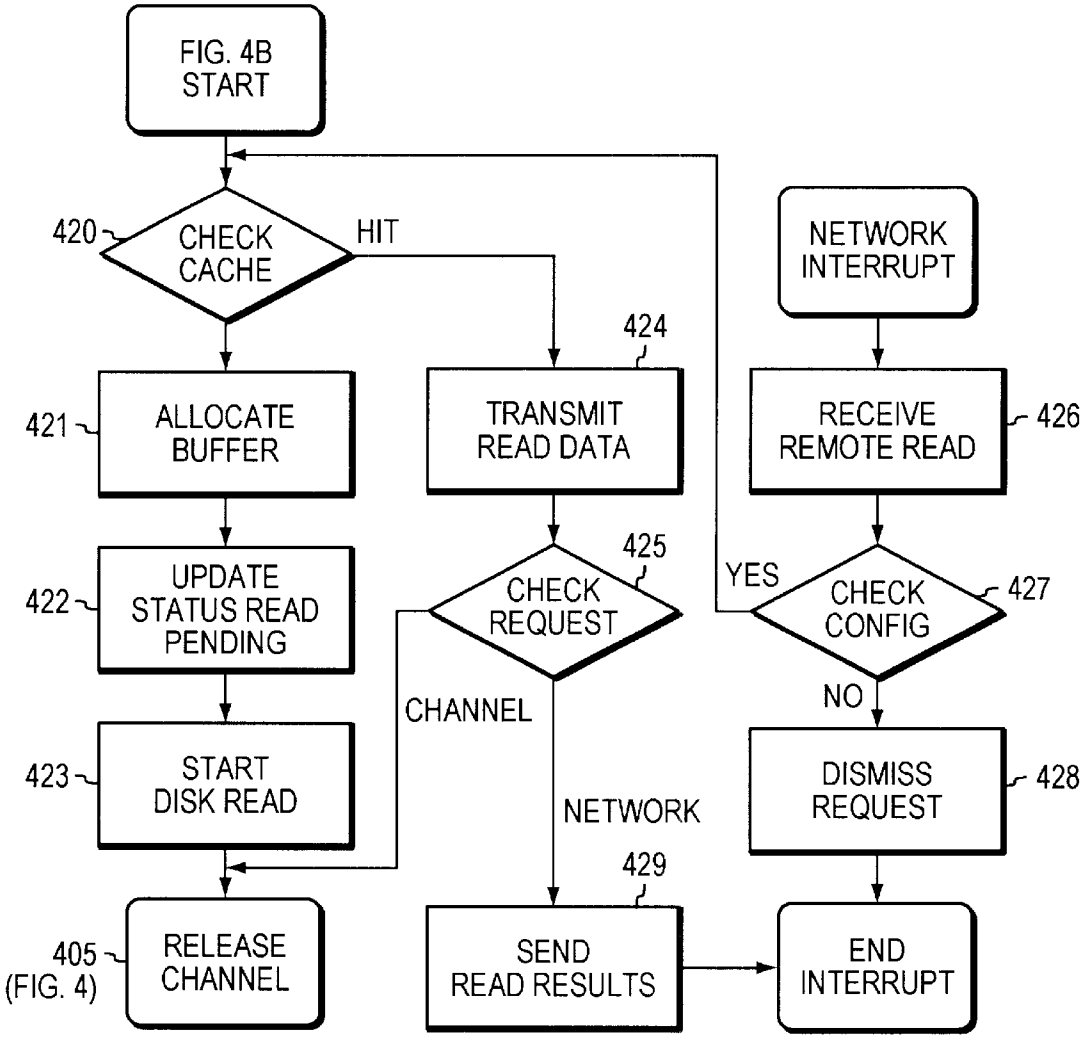


FIG. 4B START
420
CHECK CACHE
HIT
421
ALLOCATE BUFFER
422
UPDATE STATUS READ PENDING
423
START DISK READ
405 (FIG. 4)
RELEASE CHANNEL
424
TRANSMIT READ DATA
425
CHECK REQUEST
CHANNEL
NETWORK
429
SEND READ RESULTS
NETWORK INTERRUPT
426
RECEIVE REMOTE READ
YES
427
CHECK CONFIG
NO
428
DISMISS REQUEST
END INTERRUPT

FIG. 4B

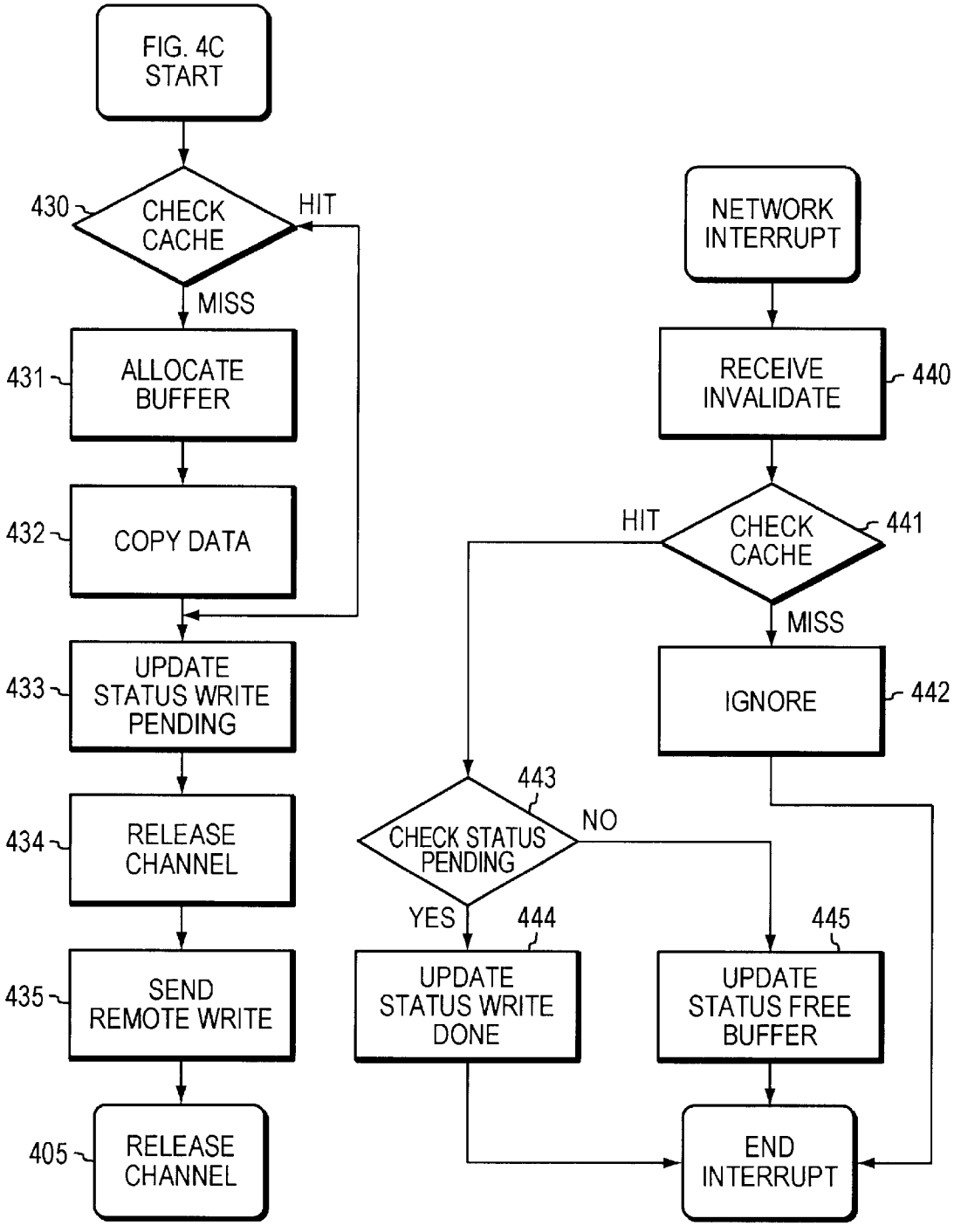


FIG. 4C START
430
CHECK CACHE
HIT
MISS
431
ALLOCATE BUFFER
432
COPY DATA
433
UPDATE STATUS WRITE PENDING
434
RELEASE CHANNEL
435
SEND REMOTE WRITE
405
RELEASE CHANNEL
NETWORK INTERRUPT
RECEIVE INVALIDATE
440
HIT
CHECK CACHE
441
MISS
IGNORE
442
443
CHECK STATUS PENDING
NO
YES
444
445
UPDATE STATUS WRITE DONE
UPDATE STATUS FREE BUFFER
END INTERRUPT

FIG. 4C

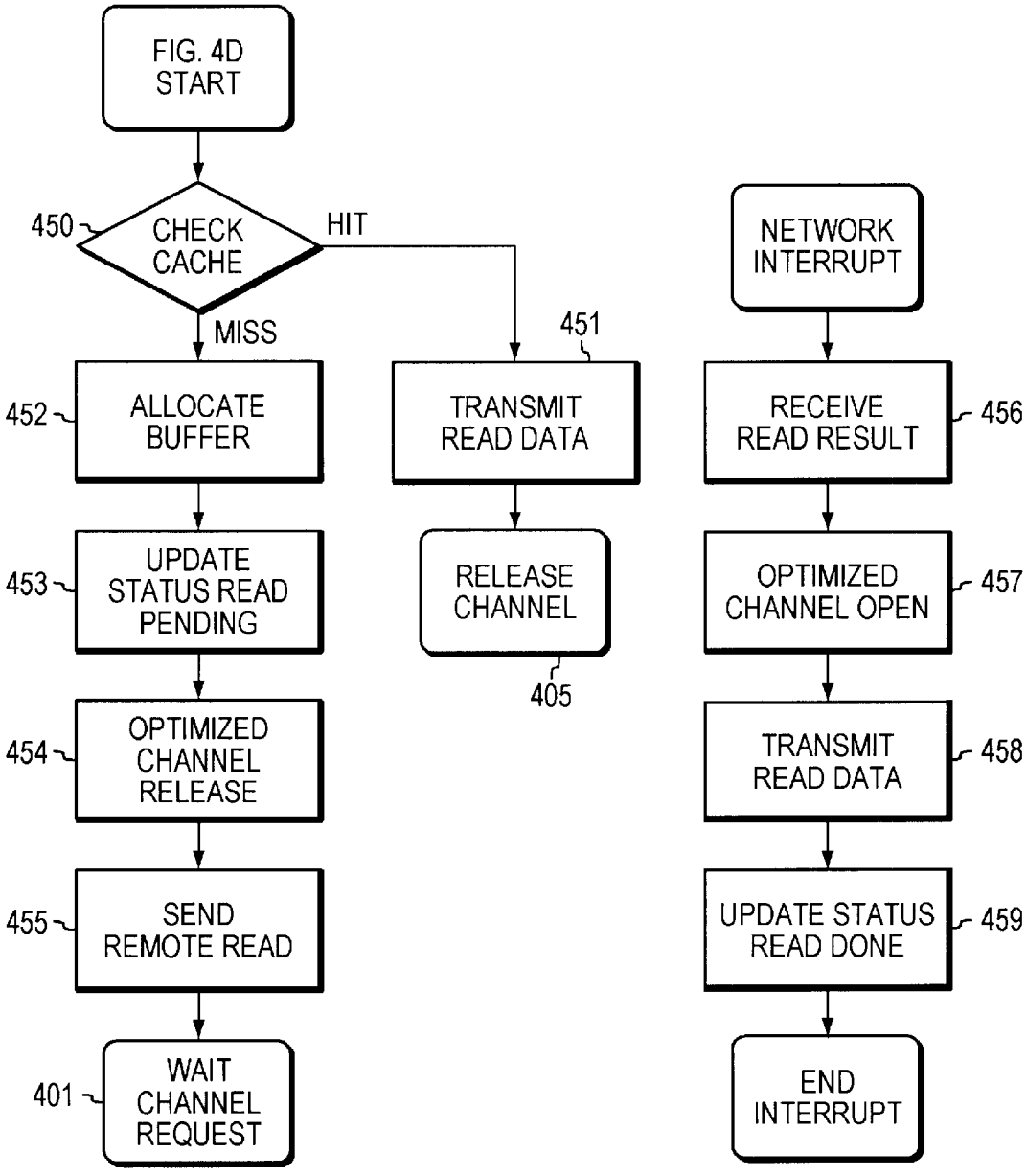


FIG. 4D START
450
CHECK CACHE
HIT
MISS
451
452
ALLOCATE BUFFER
TRANSMIT READ DATA
453
UPDATE STATUS READ PENDING
RELEASE CHANNEL
405
454
OPTIMIZED CHANNEL RELEASE
455
SEND REMOTE READ
401
WAIT CHANNEL REQUEST
NETWORK INTERRUPT
RECEIVE READ RESULT
456
OPTIMIZED CHANNEL OPEN
457
TRANSMIT READ DATA
458
UPDATE STATUS READ DONE
459
END INTERRUPT

FIG. 4D

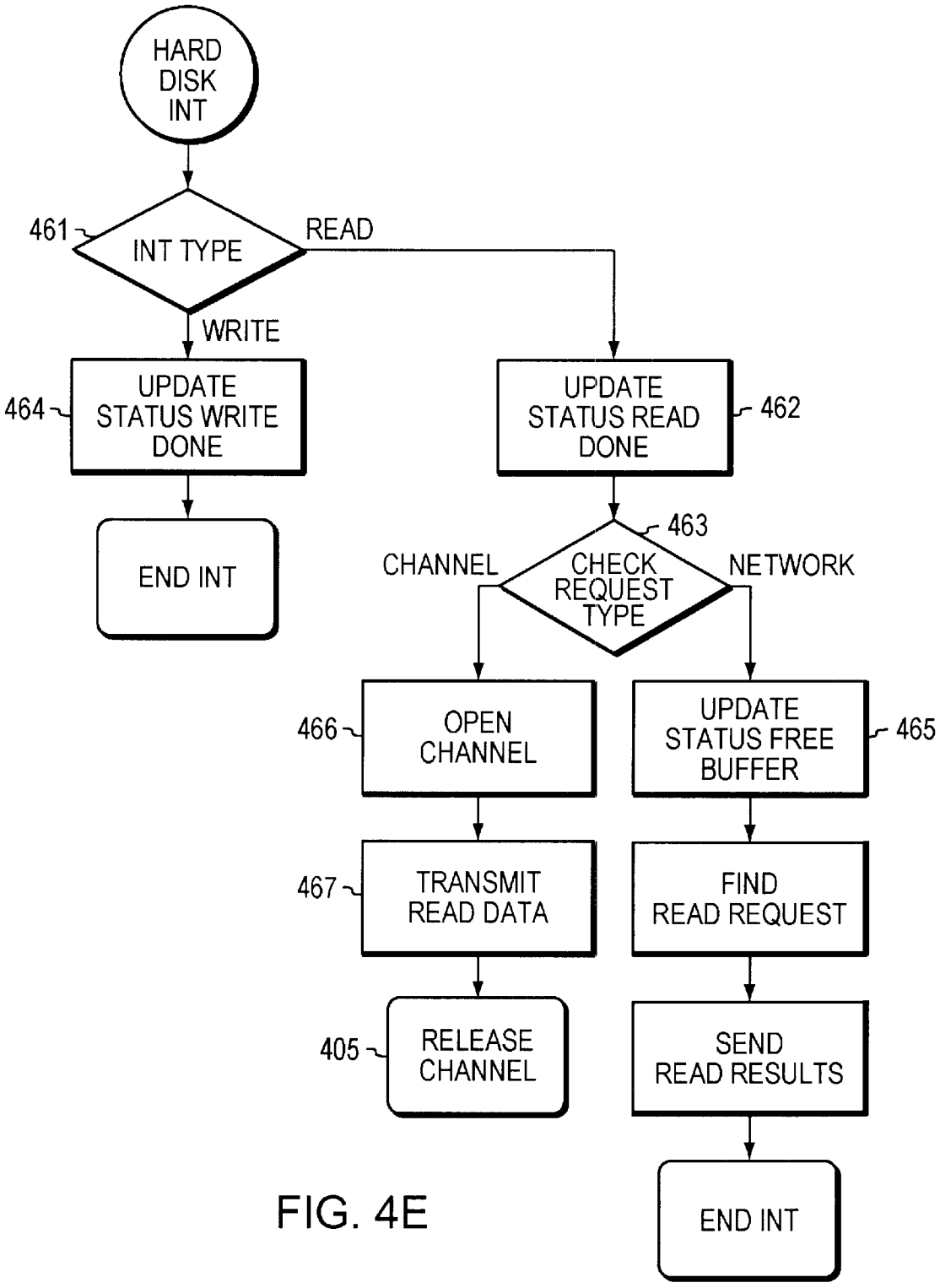

HARD DISK INT
461
INT TYPE
READ
WRITE
464
UPDATE STATUS WRITE DONE
END INT
UPDATE STATUS READ DONE
462
463
CHANNEL
CHECK REQUEST TYPE
NETWORK
466
OPEN CHANNEL
467
TRANSMIT READ DATA
405
RELEASE CHANNEL
UPDATE STATUS FREE BUFFER
465
FIND READ REQUEST
SEND READ RESULTS
END INT

FIG. 4E

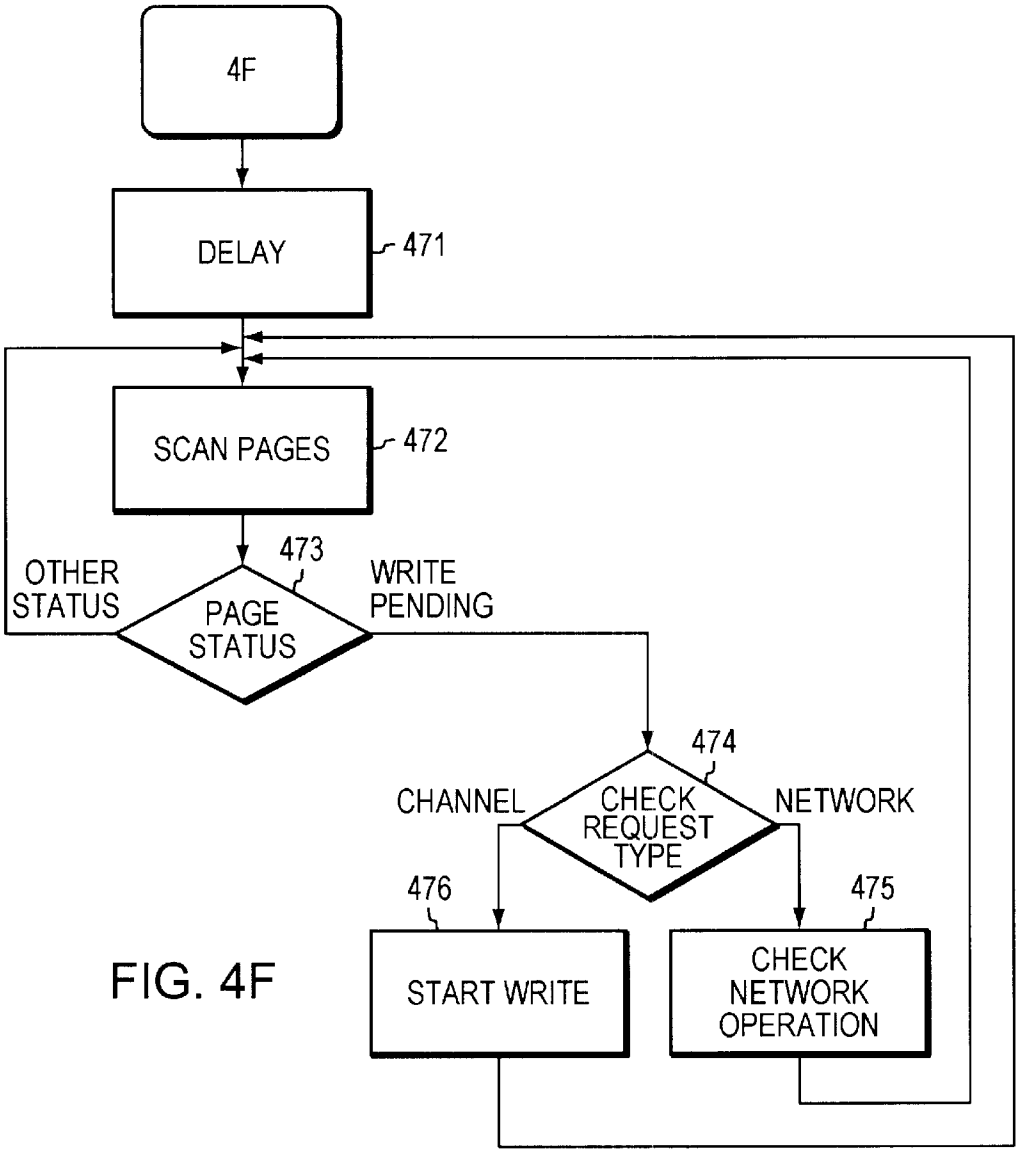

4F
DELAY
471
SCAN PAGES
472
473
OTHER STATUS
PAGE STATUS
WRITE PENDING
474
CHANNEL
CHECK REQUEST TYPE
NETWORK
476
475
START WRITE
CHECK NETWORK OPERATION

FIG. 4F

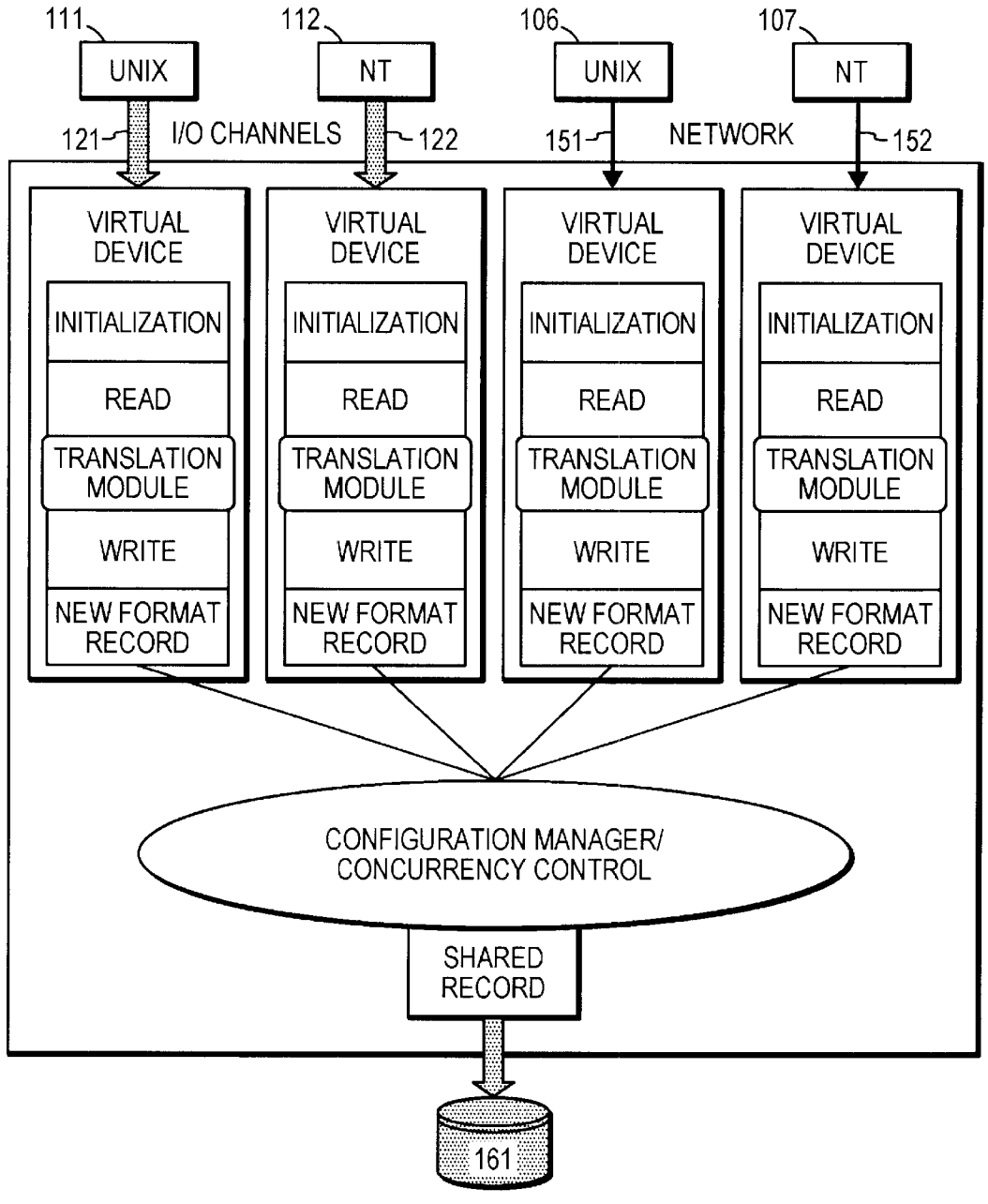

111
UNIX
112
NT
106
UNIX
107
NT
121
I/O CHANNELS
122
151
NETWORK
152
VIRTUAL DEVICE
INITIALIZATION
READ
TRANSLATION MODULE
WRITE
NEW FORMAT RECORD
VIRTUAL DEVICE
INITIALIZATION
READ
TRANSLATION MODULE
WRITE
NEW FORMAT RECORD
VIRTUAL DEVICE
INITIALIZATION
READ
TRANSLATION MODULE
WRITE
NEW FORMAT RECORD
VIRTUAL DEVICE
INITIALIZATION
READ
TRANSLATION MODULE
WRITE
NEW FORMAT RECORD
CONFIGURATION MANAGER/ CONCURRENCY CONTROL
SHARED RECORD
161

FIG. 5

DATA STORAGE SYSTEM COMPRISING A NETWORK OF PCS AND METHOD USING SAME

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates generally to the field of cached data storage systems and more particularly to a data storage system that permits independent access from local hosts connected via I/O channels and independent access from remote hosts and remote storage systems connected via network links. A network of PCs permits building a high-performance, scalable, data storage system using off-the-shelf components at reduced cost. A configuration manager ensures consistency of data stored in the distributed cache.

2. Description of Related Art

A typical data processing system generally involves a cached data storage system that connects to local host computers via I/O channels or remote host computers via network links. The purpose of the data storage system is to improve the performance of applications running on the host computer by off loading I/O processing from the host to the data storage system. The purpose of the cache memory in a data storage system is to further improve the performance of the applications by temporarily storing data buffers in the cache so that the references to those buffers can be resolved efficiently as "cache hits". Reading data from a cache is an order of magnitude faster than reading data from a back end storage device such as a disk. Writing data to a cache is also an order of magnitude faster than writing to a disk. All writes are cache hits because data is simply copied into cache buffers that are later flushed to disks.

Prior art data storage systems are implemented using proprietary hardware and very low-level software, frequently referred to as microcode, resulting in expensive and not portable systems. In contrast to the prior art systems, the preferred embodiment of the present invention uses standard hardware and software components. A network of commercial PCs is used to implement a high-performance data storage system. A method using the network of PCs includes an algorithm for a configuration manager that manages access to the distributed cache memory stored in PCs interconnected by the network.

Numerous prior art systems and methods exist for managing cache memory in a data storage system. The prior art has suggested several methods for managing cache for channel attached hosts. U.S. Pat. No. 5,717,884, Gzym, et. al., Feb. 2, 1996, Method and Apparatus for Cache Management, discloses data structures and algorithms that use a plurality of slots, each of which is used to store data files. U.S. Pat. No. 5,757,473, Vishlitzky, et. al., Cache Management system using time stamping for replacement queue, Jul. 28, 1998, discloses a method that uses time stamps to manage queues in a cached data storage system. U.S. Pat. No. 5,751,993, Ofek, et. al., May 12, 1998, Cache Management Systems, discloses yet another aspect in queue management algorithms. U.S. Pat. No. 5,600,817, Macon Jr., et. al., Feb. 4, 1997, Asynchronous read-ahead disk caching using multiple disk I/O processes and dynamically variable prefetch length, discloses read-ahead methods in cached storage systems. U.S. Pat. No. 5,758,050, Brady, et. al., May 26, 1998, Reconfigurable data storage system, discloses a method for reconfiguring a data storage system.

However, the above systems use very specialized embedded operating systems and custom programming in a very low-level programming language such as assembler. The obvious drawback of the above systems is high cost because assembler-level programming is very time consuming. Another drawback is inflexibility and lack of functionality. For example, some features such as reconfigurability in data storage are very limited in proprietary embedded systems when compared to general purpose operating systems. Finally, networking support is very expensive and limited because it relies on dedicated communication links such as T1, T3 and ESCON.

One prior art system using networking of data storage systems is disclosed in U.S. Pat. No. 5,742,792, Yanai, et. al., Apr. 21, 1998, Remote Data Mirroring. This patent discloses a primary data storage system providing storage services to a primary host and a secondary data storage system providing services to a secondary host. The primary storage system sends all writes to the secondary storage system via IBM ESCON, or optionally via T1 or T3 communications link. The secondary data storage system provides a backup copy of the primary storage system. Another prior art system is disclosed in U.S. Pat. No. 5,852,715, Raz, et al., Dec. 22, 1998, System for currently updating database by one host and reading the database by different host for the purpose of implementing decision support functions.

However, the above systems use dedicated communication links that are very expensive when compared to modern networking technology. Furthermore, the data management model is limited to the primary-node sending messages to the secondary node scenario. This model does not support arbitrary read and write requests in a distributed data storage system.

There is a growing demand for distributed data storage systems. In response to this demand some prior art systems have evolved into complex assemblies of two systems, one proprietary a data storage system and the other an open networking server. One such system is described in a white paper on a company web site on Internet. The industry white paper, EMC Data Manager: A high-performance, centralized open system backup/restore solution for LAN-based and Symmetrix resident data, describes two different systems, one for network attached hosts and second for channel attached hosts. The two systems are needed because of the lack of generic networking support. In related products such as Celerra File Server, product data sheets suggest using data movers for copying data between LAN-based open system storage and channel attached storage system.

However, the above systems are built from two systems, one for handling I/O channels, and another for handling open networks. Two systems are very expensive even in minimal configuration that must include two systems.

In another branch of storage industry, network attached storage systems use network links to attach to host computers. Various methods for managing cache memory and distributed applications for network attached hosts have been described in prior art. U.S. Pat. No. 5,819,292, Hitz, et. al., Method for maintaining consistent states of a file system and for creating user-accessible read-only copies of a file system, Oct. 6, 1998, U.S. Pat. No. 5,64,751, and Burnett, et. al., Jul. 1, 1997, Distributed file system (DFS) cache management system based on file access characteristics, discloses methods for implementing distributed file systems. U.S. Pat. No. 5,649,105, Aldred, et. al., Jul. 15, 1997, Collaborative working in a network, discloses programming methods for distributed applications using file sharing. U.S. Pat. No. 5,701,516, Chen, et. al., Dec. 23. 1997, High-performance non-volatile RAM protected write cache accel-

erator system employing DMA and data transferring scheme, discloses optimization methods for network attached hosts. However, those systems support only network file systems. Those systems do not support I/O channels.

In another application of storage systems, U.S. Pat. No. 5,790,795, Hough, Aug. 4, 1998, Media server system which employs a SCSI bus and which utilizes SCSI logical units to differentiate between transfer modes, discloses a media server that supports different file systems on different SCSI channels. However the system above is limited to a video data and does not support network attached hosts. Furthermore, in storage industry papers, Data Sharing, by Neema, Storage Management Solutions, Vol. 3, No. 3, May, 1998, and another industry paper, Storage management in UNIX environments: challenges and solutions, by Jerry Hoetger, Storage Management Solutions, Vol. 3, No. 4, survey a number of approaches in commercial storage systems and data sharing However, existing storage systems are limited when applied to support multiple platform systems.

Therefore, a need exists to provide a high-performance data storage system that is assembled out of standard modules, using off-the-shelf hardware components and a standard general-purpose operating system that supports standard network software and protocols. In addition, the needs exists to provide a cached data storage system that permits independent data accesses from I/O channel attached local hosts, network attached remote hosts, and network-attached remote data storage systems.

SUMMARY OF THE INVENTION

The primary object of the invention is to provide a high performance, scalable, data storage system using off-the-shelf standard components. The preferred embodiment of the present invention comprises a network of PCs including an I/O channel adapter and network adapter and method for managing distributed cache memory stored in the plurality of PCs interconnected by the network. The use of standard PCs reduces the cost of the data storage system. The use of the network of PCs permits building large, high-performance, data storage systems.

Another object of the invention is to provide a distributed cache that supports arbitrary reads and writes arriving via I/O channels or network links, as well as a method for sharing data between two or more heterogeneous host computers using different data formats and connected to a data storage system. The method includes a translation module that inputs a record in a format compatible with the first host and stores the translated record in a data format compatible with the second host. Sharing of data in one format and having a translation module permitting representations in different formats in cache memory provides a means for improving performance of I/O requests and saving disk storage space.

In accordance with a preferred embodiment of the invention, a data storage system comprises a network of PCs each of which includes a cache memory, an I/O channel adapter for transmitting data over the channel and a network adapter for transmitting data and control signals over the network. In one embodiment, a method for managing resources in a cache memory ensures consistency of data stored in the distributed cache. In another embodiment, a method for sharing data between two or more heterogeneous hosts includes the steps of: reading a record in a format compatible with one computer; identifying a translation module associated with the second computer; translating the record into the format compatible with the second computer and writing said translated record into a cache memory.

The preferred embodiment of the present invention involves a method for building a data storage system that provides superior functionality at lower cost when compared to prior art systems. The superior functionality is achieved by using an underlying general-purpose operating system to provide utilities for managing storage devices, backing data, troubleshooting storage devices and performance monitoring. The lower cost is achieved by relying on standard components. Furthermore, the preferred embodiment of the present invention overcomes the limitations of prior art systems by providing concurrent access for both I/O channel attached hosts and network link attached hosts.

The preferred embodiment of this invention uses SCSI channels to connect to local hosts and uses standard network links card such as Ethernet, or ATM to connect to remote hosts. The alternate embodiment of the present invention uses fiber channel link such as Fibre Channel as defined by the Fibre Channel Association, FCA, 2570 West E1 Camino Real, Ste. 304, Mountain View, Calif. 94040-1313 or SSA as defined SSA Industry Association, DEPT H65/B-013 5600 Cottle Road, San Jose, Calif. 95193. Prior art systems such as U.S. Pat. No. 5,841,997, Bleiwess, et. al., Nov. 24, 1998, Apparatus for effecting port switching of fibre channel loops, and U.S. Pat. No. 5,828,475, Bennett, et. al., Oct. 27, 1998, Bypass switching and messaging mechanism for providing intermix fiber optic switch using a bypass bus and buffer, disclosure methods that connects disks and controllers. However, the problems remain in software, solution of which require methods described in the preferred embodiment of the present invention.

The drawings constitute a part of this specification and include exemplary embodiments to the invention, which may be embodied in various forms.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows data storage systems configurations,

FIG. **2** illustrates in block diagram form the alternate embodiment of the data storage system of the present invention;

FIG. **2**A illustrates in block diagram form the alternate embodiment of the data storage system of the present invention,

FIG. **2**B illustrates in block diagram form another variation of the alternate embodiment of the present invention;

FIG. **3** shows a PC data storage system;

FIG. **4** illustrates in data flow diagram form the operations of a data storage system including: FIG. **4**A illustrating operations in write exclusive mode, FIG. **4**B in read exclusive mode, FIG. **4**C in write shared mode, FIG. **4**D in read shared mode, FIG. **4**E in disk interrupt, FIG. **4**F in page flusher; and

FIG. **5** illustrates in block diagram form data sharing operations.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Detailed descriptions of the preferred embodiment are provided herein. It is to be understood, however, that the present invention may be embodied in various forms. Therefore, specific details disclosed herein are not to be interpreted as limiting.

FIG. **1** illustrates data storage system configurations of the preferred embodiment. The PC data storage system **131**

services a plurality of channel attached host processors **111**, **112** using channels **121**, **122**, and a plurality of network attached host processors **106**, **107** using network link **151**, and a plurality of network attached data storage systems **132**, **133** using network links **152**, **153**. PC storage system **132** services channel attached hosts **157**, **158**.

Hosts **157** and **158** access a data storage system **131** indirectly via network attached data storage system **132**, thereby off loading communications protocol overhead from remote hosts **157**, **158**. Hosts **106** and **107** directly access storage system **131** via network link **151** thereby incurring communications protocol overhead on hosts **106**, **107** and therefore decreasing performance of applications running on said hosts.

Host **111** accesses remote disk **181** via local data storage system **131**, network link **153**, and remote data storage system **133** without incurring protocol overhead on host **111**. Host **157** accesses disk **161** via data storage system **133**, network link **152**, and data storage system **131** without incurring protocol overhead on host **157**. Host **106** directly accesses local disk **161** via network link **151** thereby incurring protocol overhead. The disks **191**, **192** that are attached to hosts **106**, **107** without a data storage system, cannot be accessed by outside hosts.

The preferred embodiment of the present inventions uses well-established technologies such as SCSI channels for I/O traffic and Ethernet link for network traffic. In FIG. **2**, the alternate embodiment of the present invention uses fiber channel technology for both I/O traffic and network traffic. The fiber channel connects computers and hard disks into one logical network. In one variation of the alternate embodiment in FIG. **2**, the fiber optics link is organized as a Fiber Channel Arbitrated Loop (FCAL). In another variation shown in FIG. **2**A, the fiber optics link is organized as a switching network. In yet another variation in FIG. **2**B, the fiber channel is organized in two FCAL loops connected via switch.

FIG. **3** shows a software architecture and modules of a PC data storage system corresponding to the data storage system **131** in FIG. **1**. Data is received from the hosts **111**, **112** via I/O channels **121**, **122** in front-end software module **310** in FIG. **3**. The front-end module **310** handles channel commands and places the results in cache memory **322** in the form of new data or modification to data.already stored on the disk **161**. The cache manager software module **320** calls routines in the configuration manager **340** to ensure consistency of the cache memory in other network attached data storage systems. At some later point in time, the back-end software module **342** invokes a page flusher module to write modified data to disks **161** and **162** and free up cache memory.

In FIG. **3**, front-end module **310** including I/O adapter driver software has been modified to accept target SCSI I/O requests from host **111** and **112**. Said front-end module handles I/O requests in such a manner wherein hosts **111** and **112** are not aware of a data storage systems. Hosts **111** and **112** issue I/O requests as if it's going to a standard disk.

The presence of fast access cache memory permits front end channels and network links to operate completely independent of the back-end physical disk devices. Because of this front,end/back-end separation, the data storage system **131** is liberated from the I/O channel and network timing dependencies. The data storage system is free to dedicate its processing resources to increase performance through more intelligent scheduling and data transfer network protocol.

FIG. **4** shows a flowchart of a data storage system in the process of reading or writing to data volumes stored on disk drives shown in FIG. **3**. The flowchart uses a volume access table **450** (see also FIG. **5**) and controlled by the configuration manager. Local operations begin in step **401** where the corresponding front-end module **310** of FIG. **3** allocates a channel and waits for I/O requests from the initiating hosts **111** or **112**. Remote operations begin in step **402**. Depending upon the status of the value in a volume access table **450** the requests are routed either as shown in FIG. **4**A for write exclusive mode, FIG. **4**B for read exclusive, FIG. **4**C for write shared or FIG. **4**D for read shared. Concurrently with the processing of I/O operations, the independent page flusher daemon shown in FIG. **4**F scans cache memory and writes buffers to disks. Disk interrupt processing is shown in FIG. **4**E.

Volume access table **(450)** in FIG. **4** contains a mapping between hosts and volumes specifying an access mode value. If the access mode is set to neither shared nor exclusive configuration manager forwards I/O requests directly to disk. In addition to the access mode said volume access table may contain other values that help to manager and improve performance of said data storage system.

In another embodiment of this application in FIG. **5**, Applicant illustrates yet another application of the volume access table including a translation module for a given host to volume mapping. The translation module is a dynamically loadable library that can be changed, compiled and linked at run-time. Applicant further specifies the translation module in (page 10, In 12).

A user of a data storage system can externally set the values and parameters in a volume access table. For each host and volume pair a user can explicitly specify the access mode value. For some applications, where data on a remote volume is accessed infrequently, the user may want to specify other than shared or exclusive in order to disable cache for the remote volume. By disabling caching, the user has entirely eliminated cache coherency traffic for said volume. In a data storage system a user or a system administrator actively monitors and changes the behavior of a cache manager by changing values in a volume access table in order to improve performance of said data storage system.

FIG. **4**A shows a flowchart of the cache manager **320** (see FIG. **3**) as it processes a write request in an exclusive mode. In step **411** of FIG. **4**A, the cache manager checks whether the requested buffer is in cache or not. For a cache miss, in step **412**, the cache manager allocates a new buffer for storing data that will be written. For a cache hit, the cache manager branches directly to step **413** where data is copied into the newly allocated buffer. In step **414**, the cache manager calls a configuration manager routine that sends an invalidate request to the list of shared hosts for this particular volume. In step **415**, the cache manager checks the type of a request. For a channel type of a request, the cache manager returns to step **405** to release the channel. For a network type of a request, the cache manager proceeds to release network request in step **419** on the right side of FIG. **4**A.

On the right side of FIG. **4**A, in step **416**, network interrupt identifies and receives a remote write request. In step **417**, the cache manager calls configuration manager routine to determine the validity of the request. Bad requests are ignored in step **418**. Correct requests proceed to step for **410** for write exclusive processing. Step **415** returns the flow to step **419**, which releases network resources.

FIG. **4**B shows a flowchart of the cache manager as it processes a read request in an exclusive mode. In step **420**, the cache manager checks whether the requested buffer is in

cache or not. For a cache miss, in step **421**, the cache manager allocates a buffer for storing data that will be read in. In step **422**, the cache manager updates the buffer status with read pending. In step **423**, the cache manager starts an operation to read from a hard disk driver and proceeds to release the channel in step **405**. For a cache hit, in step **424**, the cache manager transmits read data and proceeds to release the channel in step **405**. For an identified network request, in step **425**, the cache manager sends back read results in step **429**.

On the right side of FIG. **4**B, in step **426**, network interrupt identifies and receives a remote write request. In step **427**, the cache manager calls a configuration manager routine that checks the configuration file and ignores bad requests in step **428**. Correct requests proceed to step **420** for read exclusive processing. Step **425** returns the flow to step **429** that sends read results.

FIG. **4**C shows a flowchart of the cache manager as it processes a write request in a shared mode. In step **430**, the cache manager checks whether the requested buffer is in cache or not. For a cache miss, in step **431**, the cache manager allocates a new buffer for storing data that will be written. For a cache hit, the cache manager branches directly to step **432** where data is copied into the newly allocated buffer. In step **433**, the cache manager updates the buffer status with write pending and proceeds to step **434** to release the channel. In step **435**, the cache manager calls a configuration manager routine that sends a remote write request to the host that holds this particular volume in an exclusive mode. In follow up to step **435**, the cache manager returns to the beginning of FIG. **4**.

On the right side of FIG. **4**C, the cache manager updates the buffer status with write done in step **444**. The flow begins with the network interrupt that calls configuration manager to validate the request in step **441**. Bad requests are ignored in step **442**. A correct request proceeds to step **443** that checks whether the status of this particular buffer is write pending. If the status is pending, in step **444**, the cache manager updates the buffer status to write done. For any other buffer status, in step **445**, the cache manager updates the status to free. This buffer is released in accordance with the invalidate request that has come from a remote host that holds this volume in an exclusive mode as has been described in FIG. **4**A.

FIG. **4**D shows a flowchart of the cache manager as it processes a read request in a shared mode. In step **450**, the cache manager checks whether the requested buffer is in cache or not. For a cache miss, in step **452**, the cache manager allocates a buffer for storing data that will be read into. For a cache hit, in step **451**, the cache manager transmits read data and proceeds to step **405** to release the channel. In the case of the cache miss, the cache manager allocates a new buffer in step **452** and updates its status to read pending in step **453**. In step **454**, the cache manager closes the channel with an optimizer that maintains a pool of open channels which are kept open only for the specified amount of time. In step **455**, the cache manager calls configuration manager routine that sends a remote read request to the host that holds this particular volume in an exclusive mode. The operations of the host holding volume in read exclusive mode have been shown in FIG. **4**B.

On the right side of FIG. **4**D, in step **456**, a network interrupt identifies a remote read result. In step **457**, the cache manager performs an optimized channel open. Depending upon the status of the optimizer that has been initiated in step **454**, the cache manager may immediately get access to the still open channel or, if the optimizer fails, the cache manager may need to reopen the channel. In step **458**, the cache manager transmits read data. In step **459**, the cache manager updates the buffer status to read done and proceeds to step **459** where it releases the channel.

FIG. **4**E shows a flowchart of the cache manager as it processes a hard disk interrupt request marking the completion of a read or write request. The read request has been started in step **423** in FIG. **4**B. The write request has been started in step **475** in FIG. **4**F. In step **460**, the cache manager checks the type of the hardware interrupt. For a write interrupt in step **46 1**, the cache manager updates the buffer status to write done and releases resources associated with the interrupt. For a read interrupt in step **462**, the cache manager updates the buffer status to read done. In step **463**, the cache manager checks request type of the read operation that has been started in FIG. **4**B. For a channel request, the cache manager proceeds to open a channel in step **466**. In step **467**, the cache manager transmits read data and proceeds to release the channel in step **405**. For a network request in step **464**, the cache manager finds the remote read requests that initiated the request. In step **466**, the cache manager sends read results and ends interrupt processing.

FIG. **4**F shows a flowchart of a cache memory page flusher. The flusher is a separate daemon running as part of the cache manager. In step **471**, the flusher waits for the specified amount of time. After the delay in step **472**, the flusher begins to scan pages in cached memory. In step **473**, the flusher checks the page status. If the page list has been exhausted in branch no more pages, the flusher returns to step **471** where it waits. If the page status is other than the write pending, the flusher returns to step **472** to continue scanning for more pages. If the page status is write pending, the flusher proceeds to step **474**. In step **474**, the flusher checks the request type. For a channel type, the flusher starts a read operation in step **475** and returns to scan pages in step **472**. For a network type, the flusher checks for the network operations in progress and returns to step **472** for more pages.

FIG. **5** shows a data sharing operation between a plurality of heterogeneous host computers. In one embodiment the plurality of hosts includes but is not limited to a Sun Solaris workstation **111**, Windows NT server **112**, HP UNIX **106**, and Digital UNIX **107** each accessing a distinct virtual device respectively **510**, **520**, **530** and **540**. Configuration manager, **560** provides concurrency control for accessing virtual devices that are mapped to the same physical device **161**. The configuration manager uses a volume access table **450** that has been shown in FIG. **4**.

A virtual device is a method that comprises three operations: initialization, read and write. The initialization operation registers a virtual device in an operating system on a heterogeneous host. Following the registration, the virtual device appears as if it is another physical device that can be brought on-line, offline or mounted a file system. An application program running on the host cannot distinguish between a virtual device and a physical device.

For a virtual device, the read operation begins with a read from a physical device followed by a call to a translation module. The translation module inputs a shared record in a original format used on a physical disk and outputs the record in a new format that is specified for and is compatible with a host computer. The write operation begins with a call to a translation module that inputs a record in a new format and outputs a record in a shared format. The translation module is a dynamically loadable library that can be changed, compiled and linked at run-time.

The virtual device method described above allows a plurality of heterogeneous host computers to share one copy of data stored on a physical disk. In a data storage system using said virtual device method, a plurality of virtual devices is maintained in cache without requiring a copy of data on a physical disk.

While the invention has been described in connection with a preferred embodiment, it is not intended to limit the scope of the invention to the particular form set forth.

What is claimed is:

1. A computer suitable for use in a data storage system comprising a network interconnecting a plurality of such computers, the computer comprising:

an I/O channel adapter for accepting an incoming I/O request from a host;

configuration manager software for enabling said I/O channel adapter to decide whether (i) to route said request to cache, (ii) to route said request to disk, or (iii) to reject said request;

a network adapter for handling network control traffic;

a cache memory;

front-end software for handling I/O requests arriving at the I/O channel adapter or the network adapter;

cache manager software, responsive to said front-end software, for handling data stored in said cache memory; and

back-end software, responsive to said configuration manager software, for handling reads and writes to disks corresponding to the I/O requests but without communication over the I/O channel adapter, thereby separating disk operations from network and I/O traffic.

2. The system of claim **1** wherein the computers comprise off-the-shelf hardware and operating systems and further comprise:

an adapter I/O software for accepting incoming I/O requests from a host; and

a volume access table employed by the configuration manager to ensure consistency of data stored on the network.

3. The system of claim **1** wherein the cache memory comprises a portion of a distributed cache memory stored in the computers interconnected by the network.

4. The system of claim **3** further comprising a volume access table employed by the configuration manager to ensure consistency of data stored in the distributed cache.

5. The system of claim **4**, wherein the configuration manager includes software that checks an access mode in the volume access table and (i) if the access mode is set to an exclusive mode, causes both reads and writes to be stored in the cache memory, and causes invalidate messages to be sent to remote storage systems; (ii) if the access mode is set to shared, causes only reads to be stored in the cache memory; and (iii) if the access mode is set to a value other than exclusive or shared, causes reads and writes to be performed directly to a disk without using the cache memory.

6. A method of accessing a remote disk over a computer network without incurring network overhead, the method comprising the steps of:

a. causing a local host to issue a request over an I/O channel to a local computer;

b. providing a configuration manager on the local computer, the configuration manager routing the request to a remote computer via the computer network;

c. causing the remote computer to check the request against a volume access table;

d. causing the remote computer to perform an I/O operation on a disk located on the remote computer and to return data to the local computer;

e. causing the local computer to provide the returned data to the local host via the I/O channel; and

f. causing the local computer to check the data against the volume access table to ensure consistency of the data on the local and the remote computers.

* * * * *